AD-A086 827 NAVAL SURFACE WEAPONS CENTER DAHLGREN VA F/G 5/9
ANTICIPATING THE SOFTWARE ENGINEER: THE ACADEMIC PREPARATION.(U)
MAY 80 R E NANCE, W P WARNER
UNCLASSIFIED NSWC/TR-80-108 NL

1 OF 1
AD
A086827

END
DATE
FILMED
8-80
DTIC

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

14

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER NSWC TR-80-108	2. GOVT ACCESSION NO. AD-A086 827	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ANTICIPATING THE SOFTWARE ENGINEER: THE ACADEMIC PREPARATION		5. TYPE OF REPORT & PERIOD COVERED Final rept.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Richard E. Nance Walter P. Warner		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS NAVAL SURFACE WEAPONS CENTER (K70) DAHLGREN, VA. 22448		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NIF
11. CONTROLLING OFFICE NAME AND ADDRESS NAVAL SURFACE WEAPONS CENTER DAHLGREN, VA. 22448		12. REPORT DATE May 1980
		13. NUMBER OF PAGES 45
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) 12 34		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Beginning with the specification of the knowledge required to develop systems at the Naval Surface Weapons Center (NSWC), this report focuses on the identification of the knowledge to be gained through academic preparation and on-the-job experience. Discrimination of the relative importance of knowledge areas also enables the distinction between software and systems engineers. A review of efforts to define software engineering and to provide academic programs leads to the conclusion that NSWC requirements are not (over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

411567

atypical. Conclusions and recommendations on the development of software engineers specifically treat the issues of the definition of software engineering, the confusion between software and systems engineers, the areas of academic preparation, the knowledge background to be gained through experience, and the necessary training of management with regard to software development technology.

FOREWORD

This report was prepared at the request of the Head, Strategic Systems Department.

Released by:

ROBERT T. RYLAND, JR., Head
Strategic Systems Department

CONTENTS

ILLUSTRATIONS

TABLES

EXECUTIVE SUMMARY

REASONS FOR THE STUDY

In 1968 the late George Forsythe[1] commented on the lack of stability in the definition of computer science and the differences in perceptions of those viewing educational programs in an article entitled, "What to Do Till the Computer Scientist Comes." The authors of this report find the current outlook on the proper academic preparation in computing to reflect many of the difficulties and concerns characterized more than a decade ago by Professor Forsythe. Today's shortage is described in a different "currency"--software engineer--but the needs are expressed in very familiar terms.*

In December 1979 the authors of this report were requested to examine the needs of NSWC for personnel who possessed the capabilities to develop the software supporting the systems being produced at the Center. At the time of this request such personnel were generally called "software engineers," but the responsibilities and capabilities of software engineers were perceived quite differently, both within and outside the Center. Mr. R. T. Ryland, through his request to the authors, hoped to recognize and resolve some of these differences and to focus attention on the academic preparation of future software development personnel.

Even with the large and diverse NSWC experience in software development, the mustering of the needed personnel expertise has been largely through on-the-job rather than academic training. Typically, the software for the general-purpose computers has been developed by persons having a mathematics or computer science academic background. In general, the software for operational systems (subsequently to be defined) has been developed by personnel having academic backgrounds in electrical engineering, mathematics, physics, and computer science. Only in recent years have personnel been available with academic backgrounds in computer science. Academic programs in software engineering are only now emerging and have yet to produce graduates.

The study described in this report began with a definition of the knowledge needed to cope with the systems and software tasks in the Center. An examination of the manner in which universities and professional societies viewed the software engineer provided a reference point.

---

* Interestingly, one of the authors (Nance) first heard the term "software engineer" in a conversation with Professor Forsythe in 1969.

CONCLUSIONS

1. The definition of software engineering given by Bauer[2] is appropriate in general and also descriptive of the skills needed for accomplishing the Center's tasks.

2. Software engineering is sufficiently recognized to be considered an area of professional responsibility.

3. We concur with the perception that a graduate program in software engineering is more appropriate, considering both the general requirements and the Center's needs, than an undergraduate program.

4. A general misconception of the software engineer exists within the Center stemming from the historical manner of developing systems, a confusion in the roles of software and systems engineers, and a lack of appreciation for the importance of software development technology.

5. The definition of software engineer produced by the Navy Study Group differs significantly from that accepted by those dealing with the academic preparation.

6. The basic requirements identified by the Navy Study Group concerning Civil Service classification reflect a preoccupation with engineering as opposed to computer science, mathematical sciences, and information systems for academic preparation in software engineering.

7. By identifying the knowledge areas contributing to the education and training of software engineers, the distinction between software and systems engineering is clarified.

8. Academic preparation and on-the-job experience are both essential to the development of software engineers.

9. Academic preparation in software engineering for Center personnel should include interpersonal communication skills, functional capability of digital hardware, software design technology, programming systems techniques, and information structures as primary areas of study.

10. Secondary areas of academic preparation should include process exposure, design principles, systems integration, human factors engineering, and systems simulation.

11. On-the-job experience can serve better than academic preparation for Center personnel in the knowledge areas: process exposure, design principles, software design technology, and systems integration.

12. The training of software engineers for operational systems should be consciously structured to provide more exposure to and interaction with hardware and total system problems.

13. Experience is a major contributor to the training of systems engineers, perhaps playing a more significant role than the background academic discipline(s).

RECOMMENDATIONS

1. The Center should adopt Bauer's definition of software engineering:

> the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

2. The Center should recognize that the work of the IEEE Software Engineering Subcommittee supports the academic preparation of software engineers conforming to the above definition and responsive to the Center's needs.

3. The Center should consider the Master's program to be the appropriate level for the academic preparation of software engineers.

4. Through its two existing committees, the Center should seek to clarify the relationship between software engineering and systems engineering.

5. The Center should determine ways to educate management regarding the software development technology and its importance to the Center's products.

6. The Center should attempt to influence the definition of the position of software engineer put forth by the Navy Study Group.[3]

7. The Center should identify those academic programs that (1) provide the primary areas of study and (2) offer work in the secondary areas.

8. The Center should closely examine the training programs and early assignments of new hires so as to include those essential knowledge areas gained on the job in the experience of prospective software engineers. Special attention should be given to those intended to work in the operational systems environment to introduce them to the additional requirements posed by management practices, the variations in hardware interfaces, and the crucial need for a systems perspective.

9. The Center should investigate the possibility of retraining for developing software engineers by examining the prior experiences of employees to assess their potential for changing career paths.

Articulation

Immigration Course: A two-week orientation with emphasis on the problem areas of SoftE and the program objectives. Local computing facilities are introduced.

Software Methodology

Software Methodology I,II: Software lifecycle concepts; systems analysis, requirements definition, architectural and detailed design, structured programming, programming style, concurrent programming, abstract data types, formal verification, testing, validation and verification, maintenance; reviews, project structure, and cost estimation.

Software Systems Architecture: Anatomy of software systems: scheduling, resource allocation, data and file management, telecommunications; compilers, editors, loaders; run-time support packages. Interface considerations--effects of performance, modifiability, testability. Layering of abstract machines. Notations for systems representations.

Management of Software Development

Software Business I,II: Fundamentals of economics and cost accounting, foundations of organizational behavior, management of innovation. Project management, milestones. Software cost estimation. Overview of the computer industry.

Formal Methods

Use of relations, functions, and graphs in data management,structured programming and flow analysis. Applications of regular languages, computability theory, and finite state automata to the language translation problem. Techniques of algorithm analysis and complexity theory.

Software Development Laboratory

Project I,II: Simulation of the software development environment: team techniques, communication skills, reporting, planning, specification, and documentation of a significant programming project.

Electives

The elective courses should provide in-depth specialization in some aspect of computer science such as operating systems, compilers, database management, or telecommunications; or perhaps in some application area such as business data processing, scientific programming, real-time systems, computer graphics, or medical computing.

Figure 1. Composition of a 12-Month Master's Program

VIEWS OF SOFTWARE ENGINEERING

GRADUATE EDUCATION IN SOFTWARE ENGINEERING

Graduate programs in software engineering (SoftE) exist in a few colleges and universities. These programs are recent additions and as a consequence some differences are to be expected. However, the agreement between two existing Master's programs is striking, with the differences appearing in the packaging of subject content and the relative emphasis on business topics (economics, cost accounting, and organizational behavior) versus programming techniques.[4,5] The emergence of consensus is also apparent in the efforts of the Software Engineering Subcommittee (SES) of the IEEE Computer Society Education Committee. The SES, chaired by Professor Richard E. Fairley, is converging on the definition of a model curriculum that represents an "ideal" and guide for academic program development.

Five foundation areas are identified by Fairley and Wasserman (Reference 6, pg. 31) as common to graduate SoftE programs: computer science, management science, communications skills, problem solving, and design. These five areas, first proposed by Freeman, Wasserman, and Fairley,[7] have persisted as components of the SES "ideal,"[8] and they surface noticeably in the extant Master's programs. The course structure and sequencing to realize an implementation of the foundational skill are shown in Figure 1, taken from Reference 9 with grouping by subject area and the condensing of some course descriptions. The proposed program, requiring the completion of 32 semester credit hours, could be accomplished by the sequence shown in Figure 2 (Reference 9, pg. 72).

<u>First Semester</u>

Software Methodology I
Software Systems Architecture
Software Business I
Elective

<u>Second Semester</u>

Software Methodology II
Software Business II
Formal Methods
Elective
Project

<u>Summer</u>

Elective
Project II

Figure 2. Scheduling for a 12-Month Graduate Program
(All courses are three semester credit hours.)

UNDERGRADUATE EDUCATION IN SOFTWARE ENGINEERING

A four-year undergraduate program in SoftE has been proposed by Jensen, Tonies, and Fletcher[10]. While differing significantly from the graduate program proposed above, the foundational areas (computer science and engineering, management science, communications skills, and problem solving) are almost identical. The proposed curriculum in a course format is shown in Appendix B.

Fairley and Wasserman in a recent survey of proposed educational programs in SoftE note that "minimum preparation for a graduate program would be modeled after existing undergraduate programs in computer science."[6] They conclude that "the ideal entrant into the program would have undergraduate training in computer science plus two years' work experience with large software systems." Also significant in this paper is the comment on the retraining or continuing education in SoftE:

> To be effective in the work environment, retraining must be supported by new management policies, new standards, and new operating procedures which reinforce the new methods and techniques adopted.

A NAVY VIEW OF SOFTWARE ENGINEERING

In August 1979 a Computer Software Engineer Study was initiated through the office of the Chief of Naval Operations.[11] This study sought to define the specific engineering functions performed by a software engineer and to differentiate proposed positions from the existing GS-334 (computer specialist) and GS-1550 (computer scientist) series. Among the several conclusions of this study group, the following are most significant:

1. Differences in the concept of "software engineer" exist among the various activities.

2. The "ideal" software engineer should have an engineering undergraduate degree and a graduate computer science degree, or equivalent, followed by unique training such as in defense acquisition management.

3. It was proposed that software engineers be classified only at the GS-12 and above grade level.

In the reporting memorandum for the meeting[3], the study group presented its definition of "computer software engineer" as

> .. a professional engineer responsible for various aspects of software system design, development, and management essential to ensure effective utilization of computer system resources as elements of major physical or environmental systems which incorporate one or more specific engineering disciplines. The

> computer systems are generally embedded and integrated within a major system complex and provide direct real-time support of and/or perform specific tasks within one or more of the system functional elements.

The basic requirements for a computer software engineer were specified as the following:

1. Successful completion of four-year professional engineering or scientific curriculum leading to a bachelor's or higher degree in an accredited college or university

2. Inclusion of differential and integral calculus and courses in the following areas of computer and engineering sciences: digital devices, programming languages, operating systems, computer systems architecture, finite and discrete mathematics, mathematical modeling, compilers, and software engineering.

SUMMARY

The content and structure of the curriculum proposed by Fairley[8] (and presumably under consideration by the SES) appear to be sound and balanced. For NSWC purposes the two-course sequence in software business seems excessive. Possibly, one course would be sufficient. We find ourselves in agreement with Fairley and Wasserman: "... that there is simply too much material and experience for a student to absorb in a four year undergraduate program."[6]

The following statement from Fairley and Wasserman[6] is a candid expression of concern with the "what's in a name" syndrome:

> It is unfortunate that the name "software engineering" implies implementation of the program by an engineering department. There are at least three departments in most universities that have a legitimate interest in teaching software engineering: the computer science department, the electrical engineering department, and the management information systems department. We foresee that the body of material will be taught under many different names, in many different departments.

The authors (Fairley and Wasserman), who have been intimately involved with the IEEE Computer Society educational activities, are to be commended for their catholic perspectives.

The differences in perceptions of software engineering currently reflected by the Navy Study Group are significant. These differences are manifested in the emphasis on engineering, the restriction to "embedded and integrated systems," and the requirement of "direct real-time support." The efforts

of this group are still in progress, and hopefully their subsequent work will reflect a broader position closer to that of the curriculum definition of the IEEE Computer Society Software Engineering Subcommittee.

NSWC BACKGROUND IN SOFTWARE DEVELOPMENT

SOFTWARE DEVELOPMENT PROJECTS

The history of software development at the Naval Surface Weapons Center began with the advent of large digital computers for laboratory usage in the late 1940s. Because of its longstanding mission responsibility for the numerical data required to aim, target, or control Navy weapons, it was the first naval activity to have a large-scale computer. Correspondingly, the Center was the first navy activity to develop and support software for Navy deployed operational digital systems, beginning about 1960.* With the continually evolving Navy requirements for digital computer applications, the Center has sustained its leading role in digital computer applications and software development expertise.

The Center is responsible for the complete weapons control software package development, testing, and operational support for the Fleet Ballistic Missile System: POLARIS, POSEIDON, and TRIDENT. It has provided the wrap-around simulations and facilities for testing the digital fire control programs for the TARTAR, TERRIER, and TALOS surface missile systems. Similar support has been provided for the Mk 86 and Mk 92 gunfire control systems. In connection with the Navy's Gunnery Improvement Program, the Center has developed the software for the Mk 68 digital fire control system for 5-inch guns.

The Center has pioneered the application of minicomputers for Fleet electronic warfare (EW) systems and ELINT processing systems. These digital systems have enabled orders of magnitude advancements in processing quality and quantity, in data response time, and in overall Fleet EW effectiveness. Two current major examples are the development of the Airborne ESM Data Analysis Systems and the support of the AN/SLQ-32 EW Countermeasures Suite. The Intelligence Analysis Center for the Marine Air/Ground Intelligence System (MAGIS) and the shipboard Intelligence Center for LHA and CVV installation are additional examples of systems developed by the Center. These are the first major deployed intelligence database oriented systems, both incorporating Navy standard computers. Among the large software systems developed on general purpose computers are the TRIDENT Advanced Weapon System Simulation and CELEST (a satellite orbit determination program).

Operational software might also be described as software for "embedded computer systems"[13]. The principles and techniques underlying the software

---

* Following the terminology used by the Navy Laboratory Computing Committee[12], we use the term "operational software" to describe these programs.

development task for large, complex systems apply regardless of the applications context. However the enclosing system and the application often impose constraints and limitations of time (real-time requirements) and storage that are shared only by the most challenging general purpose programs (e.g.,--operating systems, run-time control systems, etc.). The Center estimated that 400 personnel were engaged in the support of operational software alone in 1975 (Reference 12, pg. D-1).

PERCEPTIONS OF SOFTWARE DEVELOPMENT AND SOFTWARE ENGINEERING

During the initial Committee meeting and throughout the deliberations, we identified several perspectives on software engineering that appeared to be created by the nature of Navy systems or the particular historical environment of the Center. To some extent these perspectives can be recognized in the conclusions of the Navy Study Group. Our treatment of these biases was to place them squarely in view, define them as precisely as possible, and argue as to their credibility, generality, accuracy, and acceptability. This straightforward manner helped us to learn more about the needs and requirements for software development at NSWC.

We note the significant biases that surfaced during our meetings so as to furnish a more complete understanding of the conclusions and recommendations in this report.

> <u>"Software Last"</u>--This bias stems from the tradition of designing and sometimes developing the hardware subsystem, then forcing the software to deal with the undocumented, changing interfaces. This unfortunate tradition has led to the perception that the software designer/developer/implementor must have sufficient knowledge of the other subsystems to deal with <u>all</u> the interface problems.
>
> Actually, the software subsystem should be designed in the same time frame as the other subsystems and no greater responsibility placed on the design of the software subsystem than any other.
>
> <u>SMOP (Simple Matter of Programming)</u>--The view that software development is a relatively trivial matter undeserving of detailed definition, planning, and control is documented in Reference 12 (pg. C-1). A consequence of this view is that the software, which can often prove to be the pacing item, is developed at excessive cost and rarely is completed on time.
>
> Critical to the software development task is the detailed definition of requirements. This must be included in the systems-level planning and the design process must be treated as prominently as that for hardware.

"Software or Systems?"--Because of the "software last" problem, the perception was created that the software designer/developer/implementor had to solve the problems of varying, undocumented, and unknown interfaces. Consequently, some managers perceive the software task as requiring extensive and in-depth knowledge of all other subsystems. This perception eliminates the need for a systems engineer by forcing the difficult interface problems to be solved by the software engineer within a set of predefined constraints imposed by decisions already made for the other subsystems.

The proper approach to total systems design provides that the software and hardware interfaces are defined in detail at the beginning. The software is designed concurrently with the hardware, and changes in the interfaces are decided by the systems engineer after he considers the effects on the total system performance.

Operational versus General Purpose--The view that operational software is radically different from the software developed for general purpose systems is held by some.

We believe that the perceived differences in software development stem from the real differences in constraints on the software and the requirements of interfacing with different types of subsystems. No fundamental differences exist between embedded and general-purpose computers. Similarly, the concepts and principles for software development are invariant, whether applied to operational software or general purpose software. Development of operational software is enhanced by knowledge differing from that for general-purpose computers, but the reverse is equally true. In the former instance the limitations of memory or the exigencies of time-critical response prove binding, while the idiosyncrasies of a huge operating system complicate the latter. Although we hold the view that there are no fundamental differences in operational and general-purpose software development, we do maintain that management practices and implementation logistics generally place a greater strain on the operational software development task. The most beneficial training assignments and experience might prove quite different for this reason.

IDENTIFYING THE ACADEMIC PREPARATION

NEEDS SPECIFICATION

Because of the biases and differences in backgrounds of the authors, the early decision was to ignore the title of positions and the job descriptions. We began by identifying the knowledge areas important to the development of the Navy systems for which NSWC has, or could have, responsibility. These knowledge areas, considered to be necessary for those individuals developing successful systems, are

1. Controls--controls, information feedback systems, basic systems distinctions (open loop, closed loop, hierarchical, etc.).

2. Process exposure/dynamic interrelationships--time-dependent behavior, system interactions, the "process" concept, cross effects, binding time, process communication, cooperation, and competition.

3. Design principles--the principles of engineering (or the scientific method), elements of the design activity (specification, analysis, decomposition, synthesis, testing), maintenance, and reliability.

4. Interpersonal communication skills--written and verbal communication, team participation, and team leadership.

5. Functional capabilities of digital hardware--logical structure and composition. (This area was recognized to be potentially divisible into computer hardware and digital non-computer hardware.)

6. Software design technology--system life cycle, specification techniques (e.g., PSL/PSA, Workbook, Jackson, etc.), development techniques (e.g., chief programmer, structured walk-through, design reviews, builds, code reading), documentation, modification, and maintenance (see Jensen and Tonies[14]).

7. Evaluation--systems analysis techniques, models and modeling, identification or creation of alternatives, characterization of trade-offs.

8. Systems integration--component and subsystem testing, systems reliability, progressive testing, diagnostic capability, degraded mode options, recovery.

9. Programming systems techniques--programming languages, systems programs, structured programming, modularity, stubs, program documentation, program testing.

10. Human factors engineering--human/machine interface, dialogue design, prompting, "trainability" and "learnability," adaptability and design for change. (This area was recognized as potentially divisible into software design for human use and hardware design for human use.)

11. Information structures--logical and physical organization of data, data definition, abstract data types, data-base technology.

12. Communications technology--digital communications, devices, data transmission, coding techniques, protocols, security.

13. Systems simulation--experimentation and system testing using simulation, discrete event and continuous simulation models, wrap-around simulators, emulation.

AREAS OF ACADEMIC PREPARATION

After reviewing several sources (References 4, 5, 8, 9, 10, 15), we recognized that a consensus exists as to the definition of software engineering (and, consequently some direction was provided in defining a "software engineer"):

> the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines (Reference 2, pg. 530).

We were pleased to discover that this definition did not differ in principle from that given in the Department of Defense Directive 5000.29 of 26 April 1976:

> (The) science of design, development, implementation, test, evaluation, and maintenance of computer software over its life cycle.

After reaching a general agreement on the definition of "systems engineer," we sought to compare or contrast the two (software and systems) using the 13 knowledge areas. Applying the scale 4--in-depth knowledge, 3--good working knowledge, 2--some knowledge, and 1--little or no knowledge, we produced a consensus estimate of the importance of each area in fulfilling the duties of the software and the systems engineer (see Table 1).

ACADEMIC OR ON-THE-JOB KNOWLEDGE

In some knowledge areas academic preparation is believed to be less effective than job experience. Table 2 reflects the consensus regarding the better source of knowledge, and in some cases a closer description of the contributing experience is given.

Table 1. Assessment of Knowledge Area Importance

Area	Software Engineer	Systems Engineer
1. Controls	2.5	4
2. Process exposure	4	4
3. Design principles	4	4
4. Communication skills	3.5	4
5. Digital hardware	3	2
6. Software design	4	2
7. Evaluation	3	4
8. Systems integration	4	4
9. Programming	4	2
10. Human factors	3	4
11. Information structures	4	2
12. Communications technology	2	2
13. Systems simulation	3	3

Table 2. Better Knowledge Source--Academic or On-the-Job

Area	Knowledge Best Provided By	
	Academic	On-the-Job
1. Controls	x	
2. Process exposure		x[a]
3. Design principles		x[b]
4. Communication skills	*	*
5. Digital hardware	x	
6. Software design	**	x[c]
7. Evaluation	x	**
8. Systems integration		x[d]
9. Programming	x	
10. Human factors	x	
11. Information structures	x	
12. Communications technology	x	
13. Systems simulation	*	*

* neither or both
** better job needed

Notes:
a. Development and support of real-time systems
b. Design of systems emphasizing hardware, software, and firmware interfaces from user needs specifications
c. Development of large software systems
d. Design and testing of systems with software and hardware components

INTEGRATION AND SUMMARY

Obviously, no academic program is likely to offer preparation in all the areas marked in the academic column of Table 2 (Areas 1,5,7,9,10,11,12,13). The identification of the most essential areas of academic preparation should follow from the integration of Tables 1 and 2. A reasonable first cut is to begin with the academic areas (those where academic preparation was judged better) designated as requiring in-depth knowledge (Area 4). This discrimination identifies only the areas of programming systems techniques (Area 9) and information structures (Area 11). Relaxing the criterion slightly admits interpersonal communication skills (Area 4), which are not clearly designated as better with academic preparation.

At this point we examine each area, judging the importance of the area regardless of the better source of knowledge. The consensus is that Areas 5 and 6 should be designated as primary knowledge, and the remaining as either "secondary" or "useful." This decision produces the resulting classification:

Primary

- Interpersonal communication skills
- Functional capabilities of digital hardware
- Software design technology
- Programming systems techniques
- Information structures

Secondary

- Process exposure/dynamic interrelationships
- Design principles
- Systems integration
- Human factors engineering
- Systems simulation

Useful

- Controls knowledge
- Evaluation
- Communications technology

We believe that an academic preparation exposing the student to the primary areas is essential. As many of the secondary areas as possible should be included.

REFERENCES

1. George E. Forsythe, "What To Do Till The Computer Scientist Comes," *American Mathematical Monthly*, Vol. 5 (May 1968), pp. 454-462.

2. F. L. Bauer, "Software Engineering," *Information Processing* 71 (1972), North Holland Publishing Co., pp. 530.

3. Memorandum for the Record from the Office of the Chief of Naval Operations, Navy Study Group, "Definition of Software Engineer," August 20, 1979.

4. A. A. J. Hoffman, Personal Correspondence, February 18, 1980.

5. Leon G. Stucki and Lawrence J. Peters, "A Software Engineering Graduate Curriculum," *Proceedings of the 1978 ACM Annual Meeting*, Washington, DC (December 4-6, 1978), pp. 63-67.

6. Richard E. Fairley and Anthony I. Wasserman, "Options in Software Engineering Education," *SIGCSE Bulletin, Vol. 10* (August 1978), pp. 31-39.

7. P. Freeman, A. I. Wasserman, and R. E. Fairley, "Essential Elements of Software Engineering Education," *Proceedings of the Second International Conference on Software Engineering*, San Francisco, CA (October 13-15, 1975), pp. 116-122.

8. Richard E. Fairley, "Educational Issues in Software Engineering," *Proceedings of the 1978 ACM Annual Meeting*, Washington, DC (December 4-6, 1978), pp. 58-62.

9. Richard E. Fairley, "Software Engineering Education," *Proceedings of the Thirteenth Hawaii International Conference on System Sciences*, Hawaii (1980), pp. 70-75.

10. Randall W. Jensen, Charles C. Tonies, and William I. Fletcher, "A Proposed 4-Year Software Engineering Curriculum," *SIGCSE Bulletin*, Vol. 10, August 1978, pp. 84-92.

11. Chief of Naval Operations, "Computer Software Engineer Study," Memorandum Ser 141C11/296134, August 20, 1979.

12. Operational Software Panel, *Report of the Operational Software Panel of the Navy Laboratory Computing Committee*, September 1975.

13. John H. Manley, "Embedded Computers--Software Cost Considerations," *Proceedings of the 1974 National Computer Conference*, Vol 43, pp. 343-347.

14. Randall W. Jensen and Charles C. Tonies, *Software Engineering* (Englewood Cliffs, New Jersey: Prentice-Hall, 1979).

15. A. A. J. Hoffman, "A Proposed Masters Degree in Software Engineering," *Proceedings of the 1978 ACM Annual Meeting*, Washington, DC (December 4-6, 1978), pp. 54-57.

APPENDIX A

THE TEXAS CHRISTIAN UNIVERSITY GRADUATE PROGRAM
IN SOFTWARE ENGINEERING

SOFTWARE ENGINEERING COURSES

Introduction

5143 Introduction to Software Engineering

Communication

6193 Effective Participation in Small Task Oriented Groups
5193 Communication Techniques in the Software Engineering Environment

Technology

6104 Overview of Computer Science
6113 Methodologies of Software Development
6123 Requirements and Specifications for Software
6133 Software Design
6142 Software Design Laboratory
7113 Software Implementation

Management/Economics

6153 Management of Software Development
6163 Economics of Software Development

Admission to the Program

Enrollment will be limited: admission to the program will be by application to and approval by a Software Engineering Admissions Committee. Both academic background and professional experience will be considered. It is expected that individuals with the following minimum qualifications will fill available spaces:

1. Admission to the Graduate School

and either

2. B.S. in Computer Science and one year of software development experience,

or

3. Substantial (i.e. three years) job related experience in software development.

Degree Requirements

A 40 semester hour program with

either

1. 34 hours of software engineering (SENG 5143, 5193, 6104, 6133, 6132, 6153, 6163, 6193, 7113) plus 6 hours of approved electives

or

2. 30 hours of software engineering (without SENG 6104) plus 10 hours of approved electives

and an oral examination.

SOFTWARE ENGINEERING (ACRONYM: SENG)
COURSE DESCRIPTIONS

SENG/COSC 5143 Introduction to Software Engineering
3 semester hours

Techniques of software design and development. The software life cycle. Methods for requirements definition, system specification, and design. Design concepts and methods. Improved programming methodologies. Methods for testing, validation, and quality control. Documentation. Software economics. Management of programming projects. Case histories.

Prerequisites: Admission to Software Engineering Program. May be taken by seniors majoring in Computer Science on a space available basis.

SENG 6193 Effective Participation in Small Task Oriented Groups
3 semester hours

Recognizing and supplying actions necessary for task oriented groups to achieve their objectives. Group maintenance roles. Group orienting roles. Task directed roles. Evaluative roles. Closure and action items. Systematic approaches to problem solving. Problem definition. Developing the solution domain. Means-end analysis. Provisions for feedback. Delineation of subproblems. Assignment of priorities. Time lines.

Prerequisities: SENG 5143

SENG 5193 Communication Techniques in the Software Engineering Environment
3 semester hours

Organization of presentation materials. Preparation of graphics for presentation. Maximizing the use of multimedia. Writing style for software documents. Development documents - requirements, specifications, design, and implementation. Technical documentation. User documentation, automated aids. Reports. Proposals.

Prerequisites: SENG 6113

SENG 6104 Overview of Computer Science
4 semester hours

Technical overview of the software engineering environment. Computer systems architecture. Software structures such as compilers, operating systems, assemblers, file systems, and data management systems. Hardware/software tradeoffs. Storage management. System support packages.

Prerequisites: SENG/COSC 5143

SENG 6123 Requirements and Specifications for Software
3 semester hours

Requirements analysis. Techniques for representing requirements. Specification development techniques. Specification languages. Automated aids. Laboratory will consist of case studies.

Prerequisites: SENG 6113

SENG 6113 Methodologies of Software Development
3 semester hours

Structured programming. Modularization. Top-down development. Levels of abstraction. Stepwise refinement. Hardware, software, and user tradeoffs.

Prerequisites: SENG 6104 or permission

SENG 6133 Software Design
3 semester hours

Design process. Major design methods--composite/structured design, data structure drive design, structural analysis, and others. Evaluation of alternate designs. Automated design aids. Design documentation.

Prerequisites: SENG 6123

SENG 6142 Software Design Laboratory
2 semester hours

Case study designs using design methods contained in SENG 6133.

Prerequisites: Should be taken in parallel with SENG 6133

SENG 7113 Software Implementation
3 semester hours

Transfer of design to code. Testing techniques. Validation. Verification. Certification. Security. Case studies.

Prerequisites: SENG 6133 and SENG 6142

SENG 6153 Management of Software Development
3 semester hours

Organization context of software development. Analysis of life cycle costs. Scheduling and budgeting techniques. Specification and control of standards for products, processes, and equipment. Personnel development and utilization. Team techniques.

Prerequisites: SENG 5143

SENG 6163 Economics of Software Development
3 semester hours

Fundamentals of economics. Distribution of costs through software life cycle. Relative hardware/software costs. Economic analysis for decisionmaking. Economic feasibility studies.

Prerequisites: SENG 6153

APPENDIX B

THE PROPOSED UNDERGRADUATE PROGRAM IN SOFTWARE ENGINEERING

Freshman Year

Fall Quarter

SE 101	Intro to Software Engineering
MATH 220	Calculus
ENGLISH	Composition and Grammar
CHEM	Chemistry (or alternative)

Winter Quarter

SE 102	Intro to Program Design
MATH 221	Calculus
ENGLISH	Composition and Grammar
PHIL 210	Deductive Logic (General Educ)
SE 102a	Intro to Program Design Lab

Spring Quarter

SE 103	Comparative Programming Langs
MATH 222	Calculus
COMM	Communications: Public, Interpers.
SE 110	Computers and Society (General Ed)
SE 103a	Comp Prog Langs Lab

Sophomore Year

Fall Quarter

SE 201	Program Design Methodologies
MATH 321	Linear Algebra
PHYS 221	Engineering Physics
ECON 200	Economics

Winter Quarter

SE 202	Program Design Methodologies II
MATH 322	Differential Equations
SE 270	Comp Arch, Assby Lang Prog
PHYS 222	Engineering Physics
GEN ED	General Education

Spring Quarter

SE 221	Comp Arch, Assby Lang Prog
SE 240	Data Structures, Algorithm Design
PHYS 223	Engineering Physics
GEN ED	General Education

Figure B-1. The Four-Year Undergraduate Program Proposed by Jensen, Tonies, and Fletcher[16]

Junior Year

Fall Quarter

SE 310	Data Base Design and Mgt
SE 315	Digital Engineering Design
BA 311	Management Concepts
ENGLISH	Engineering Reporting
GEN ED	General Education

Winter Quarter

SE 311	Data Base Design and Mgt
SE 410	Comp Sys and Data Comm
ACCT 201	Managerial Accounting
SE 316	Digital Engineering Design II

Spring Quarter

SE 350	Legal Aspects of Soft Engr
SE 411	Comp Sys and Data Comm
MATH 563	Intro to Numerical Analysis
GEN ED	General Education
TECH EL	Technical Elective

Senior Year

Fall Quarter

SE 510	Realtime Systems
SE 595	Project Lab/Seminar
TECH EL	Technical Elective
GEN ED	General Education

Winter Quarter

SE 520	Digital Control, Microcomps
SE 595	Project Lab/Seminar
TECH EL	Technical Elective
GEN ED	General Education

Spring Quarter

SE 521	Digital Control, Microcomps
SE 595	Project Lab/Seminar
SE 550	Human Factors
TECH EL	Technical Elective
GEN ED	General Education

Figure B-1. The Four-Year Undergraduate Program Proposed by Jensen, Tonies, and Fletcher[16] (Continued)

DISTRIBUTION

Commander
Naval Electronic Systems Command
Washington, DC 20360
ATTN: Code OOB-CR

Chief of Naval Operations
Department of the Navy
Washington, DC 20350
ATTN: OP-142E

Naval Ship Weapons Systems Engineering Station
Port Hueneme, CA 93041
ATTN: Code 4400
0600

DIR., Western Field Division
Naval Civilian Personnel Command
880 Front Street, Rm. 4-5-29
San Diego, CA 92188

Fleet Combat Direction Systems Support Activity
Dam Neck
Virginia Beach, VA 23461

Fleet Combat Direction Systems Support Activity
San Diego, CA 92147

Naval Research Laboratory
Washington, DC 20375
ATTN: Code 1811

Commander
Naval Data Automation Command
Washington Navy Yard
Washington, DC 20374
ATTN: Code 95

Commanding Officer
Naval Training Equipment Center
Orlando, FL 32813
ATTN: Code N214

Naval Weapons Center
China Lake, CA 93555
ATTN: Code 3108

DISTRIBUTION (Continued)

Naval Air Development Center
Johnsville, PA 18974
ATTN: Code 501

Naval Air Systems Command
Washington, DC 20360
ATTN: Code 53312

Commander
Naval Facilities Engineering Command
200 Stoval Street
Alexandria, VA 22332
ATTN: Mr. Hodges

Commander
Pacific Missile Test Center
Point Mugu, CA 93041
ATTN: Code 1201

Office of the Chief of Naval Operations
Navy Department
Washington, DC 20350
ATTN: Code OP-141C11

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314 (12)

Library of Congress
Washington, DC 20540
ATTN: Gift and Exchange Division (4)

GIDEP Operations Office
Corona, CA 91720

Professor Richard E. Fairley
Chairman, IEEE SE Subcommittee
Colorado State University
Department of Computer Science
Fort Collins, CO 80251 (3)

Professor David Rine
Chairman, IEEE Education Committee
Western Illinois University
Computer Science Department
Mecomb, IL 61455 (2)

Professor Alexander A. J. Hoffman
Texas Christian University
Computer Science Department
Fort Worth, TX 76129

Professor William F. Atchison
Chairman, ACM Education Board
University of Maryland
Department of Computer Science
College Park, MD 20742

Professor Gerald L. Engel
Chairman, ACM Curriculum on Computer Educ.
Christopher Newport College
Department of Computer Science
50 Shoe Lane
Newport News, VA 23606

Dr. John Manley
Applied Physics Laboratory/
Johns Hopkins University
Laurel, MD 20810

<u>Local</u>:

D	
E	
E30	
E41	
F	
F02	
G	
G10 (Batayte)	
K	
K02	(5)
K03	(10)
K70	(20)
N (Nance)	(10)
N21 (McConnell)	
N31	
N302 (Cullen)	
P	
P06	
P20	
R	
U	
X210	(6)
X211	(2)